УДК 681.3

М.К. Буза
Белорусский государственный университет, Беларусь
пр. Независимости, 4, г. Минск, 220030

ОБЛАЧНЫЕ ТЕХНОЛОГИИ – ОСНОВА ПРОЕКТИРОВАНИЯ ЭФФЕКТИВНЫХ ПРИЛОЖЕНИЙ

M.K. Bouza
Belarusian State University, Belarus
4, Independence Ave., Minsk, 220030

CLOUD TECHNOLOGY IS THE FOUNDATION FOR DESIGNING EFFICIENT APPLICATION

В статье исследуются основные модели облачных технологий. Сделан аргументированный выбор модели SaaS для создания Web-приложения информационного обеспечения учебного процесса. Сформировано инструментальное окружение программных средств, включая Heroku, ClearDB, MySQL SSL, Apache Maven для поддержки проектирования и функционирования приложения. Разработаны роли и функционал для каждого пользователя приложения. Определены возможности доступа к нему, изменения и пополнения.
Ключевые слова: облачные технологии**,** SaaS модель, безопасность, WEB приложение, HEROKU

The article explores the main cloud technology models. The reasoned choice of the SaaS model for creating a Web application for educational process information support has been made. A software tool environment has been formed, including Heroku, ClearDB, MySQL SSL, Apache Maven to support the design and operation of the application. Roles and functionality for each user of the application are developed. The possibilities of access to it, changes and replenishment are determined.
Keywords: cloud technologies, SaaS model, security, WEB application, HEROKU

Введение

В последние годы традиционные программные модели постепенно заменяются на интернет-модели. Разработчики последних утверждают, что в ближайшее время для облачных технологий будут созданы новые способы работы с облаками через мобильные устройства в любой точке мира. Большинство компаний могут позволить себе разворачивать свои приложения в облаках, не покупая дорогостоящего оборудования. Для этого достаточно лишь иметь доступ в интернет.

По сути, облачные вычисления это своего рода аутсорсинг компьютерных программ. Облачные технологии позволяют пользователям получать доступ к программному обеспечению и приложениям сторонних организаций и находиться в облаке вне зависимости от места нахождения пользователя, оплачивая лишь объем хранилища данных и время работы.

Данные хранятся и обрабатываются в так называемом **облаке**, которое с точки зрения клиента представляет собой один виртуальный сервер.

Облачные технологии позволяют экономить время и средства на создание необходимого программного обеспечения, предоставляя доступ к данным в любое время суток. Из существенных недостатков использования облачных технологий отметим: общая производительность при работе в облаках часто оказывается ниже чем обработка на локальных серверах и не до конца решена проблема безопасности обработки, хранения и пересылки конфиденциальных данных.

В связи с переходом Республики Беларусь на цифровую экономику актуальной стала задача перевода системы образования на цифру.

Ниже будет сформировано инструментальное окружение и представлено онлайн-приложение информационного обеспечения учебного процесса с разворачиванием его в облаке, обеспечением безопасности как доступа к нему так и его функционирования.

Хранение такого приложения в облаке и онлайн работа с ним позволит не тратить средства на вычислительные ресурсы, которых всегда не хватает в учреждениях образования.

Модели облачных вычислений

Рассмотрим три основных вида облачных вычислений.

1. SaaS(Software-as-a-Service) – программное обеспечение как услуга. Модель обслуживания, при которой подписчикам предоставляется готовое прикладное программное обеспечение, полностью обслуживаемое провайдером. Заказчикам предоставляется доступ к функциям с клиентских устройств, как правило через мобильное приложение или веб-браузер. Примерами такой модели являются Google Apps, Dropbox, Salesforce, WebEx, Concur, GoToMeeting.
2. PaaS(Platform-as-a-Service) – платформа как услуга. Здесь потребитель получает доступ к использованию информационно-технологических платформ: операционных систем, систем управления базами данных, связующему программному обеспечению, средствам разработки и тестирования, находящихся у облачного провайдера. Вся информационно-технологическая инфраструктура, включая вычислительные сети, серверы, системы хранения управляется провайдером. Он определяет набор доступных для потребителей видов платформ и набор управляемых параметров платформ, а потребителю предоставляется возможность использовать платформы, создавать их виртуальные экземпляры, устанавливать, разрабатывать, тестировать, эксплуатировать на них прикладное программное обеспечение, при этом динамически изменяя количество потребляемых вычислительных ресурсов. Основные примеры такой модели: AWS Elasctic Beanstalk, Window Azure, Heroku, Force.com, Google App Engine, Apache Stratos, OpenShift.
3. IaaS(Infrastructure-as-a-Service) – инфраструктура как услуга. Потребителю предоставляются по подписке фундаментальные информационно-технологические ресурсы – виртуальные серверы с заданной вычислительной мощностью, операционной системой (чаще всего – предустановленной провайдером из шаблона) и доступом к сетям [1]. При подписке потребитель, как правило, приобретает серверное время, а также пространство для хранения (возможно, с различной тарификацией в зависимости от производительности), заданную сетевую пропускную способность, в некоторых случаях – сетевой трафик. Примеры такой модели: DigitalOcean, Linode, Rackspace, Amazon Web Services(AWS), Cisco Metapod, Microsoft Azure, Google Compute Engine(GCE).

Анализ показал, что для разработки проектируемого приложения целесообразно использовать модель SaaS облачных технологий. Большинство приложений SaaS запускаются непосредственно через веб-браузер и не требуют каких-либо загрузок или установок на стороне клиента.

SaaS избавляет нас от необходимости загружать и устанавливать приложения на каждом отдельном компьютере. SaaS поставщики решают все потенциальные технические проблемы, связанные с хранением данных, установкой и обновлением программного обеспечения, серверами и хранилищами, значительно сокращая время и финансы, затрачиваемые на решение задачи.

Существует множество различных ситуаций, в которых модель SaaS может быть наиболее выгодной. В частности:

- начинающим или небольшим компаниям, которым нужно быстро запустить электронную коммерцию и нет времени на решение проблем по установке сервера или программного обеспечения;
- для краткосрочных проектов, требующих сотрудничества;
- при использовании приложений, которые не очень востребованы;
- для приложений, которым нужен как веб, так и мобильный доступ.

Выбор данной модели может объясняться еще и следующим:

1. SaaS модель является наиболее распространенной среди остальных моделей из-за своих возможностей и полностью раскрывает процесс работы облачных вычислений.
2. Поддержкой приложения, использующего модель SaaS, заниматься проще по сравнению с другими моделями.
3. Пользователям приложения не надо беспокоиться о необходимом оборудовании для его использования, операционном и программном окружении.
4. При создании приложения, использующего модель SaaS облачных технологий, можно воспользоваться приложениями на моделях PaaS и IaaS для развертывания приложения в облаке.

На основе данной модели разработаем приложение, для доступа к которому достаточно иметь лишь доступ в интернет.

Формирование инструментального окружения для проектирования приложения

Характерной особенностью облачных вычислений является тот факт, что потребитель может, если ему необходимо, независимо использовать вычислительные функции, такие как серверное время или сетевое хранилище данных в автоматическом режиме, без взаимодействий с персоналом поставщика услуг [2].

Чтобы поддерживать облачное приложение необходимо создание и поддержка виртуальных машин, перераспределение данных между ними, а также обеспечение безопасности информации: создание копий на разных серверах.

Среди сервисов, которые предоставляют такие услуги бесплатно укажем на Heroku, OpenShift, Google App Engine.

В качестве необходимого сервиса выберем Heroku в силу следущего. Heroku одна из первых облачных PaaS-платформ, поддерживающая ряд языков программирования: Ruby, Java, Node.js, Scala, Clojure, Python и другие. Приложение представляет собой набор исходного кода, представленного на одном из этих языков, возможно, фреймворк и некоторое описание зависимостей, которое инструктирует систему сборки относительно того, какие дополнительные зависимости необходимы для сборки и запуска приложения. Здесь не нужно вносить много изменений в приложение, чтобы запустить его в Heroku. Одним из требований является информирование платформы о том, какие части приложения работоспособны. Если мы используем какой-то установленный фреймворк, Heroku может это понять. Например, в Ruby on Rails это обычно rails-сервер, в Django это python <app> /manage.py runserver, а в Node.js это основное поле в package.json.

Heroku – это платформа, которая позволяет создавать, запускать и масштабировать приложения одинаковым образом на всех языках, используя зависимости и Procfile. Procfile предоставляет архитектурный аспект созданного приложения с возможностью масштабировать каждую часть.

Для развертывания и сборки приложения на Heroku будем использовать Git – распределенную систему контроля версий, которую многие разработчики используют для управления и контроля версий исходного кода. Платформа Heroku использует Git в качестве основного средства для развертывания приложений (есть и другие способы передачи исходного кода в Heroku, в том числе через API). При использовании Heroku API для создания и запуска приложений развертывание означает перенос приложения из локальной системы в Heroku.

Когда платформа Heroku получает исходный код приложения, она инициирует сборку исходного приложения. Механизм сборки, как правило, зависит от языка, но следует тому же шаблону, обычно извлекая указанные зависимости и создавая любые необходимые ресурсы (простые, как таблицы и сложные, как компиляция кода). Исходный код приложения вместе с извлеченными зависимостями и выходными данными фазы сборки, такими как сгенерированные ресурсы или скомпилированный код, а также язык и инфраструктура, собраны в Slug. Slug – это набор исходного кода, извлеченных зависимостей, времени выполнения языка и скомпилированных или сгенерированных выходных данных системы сборки,

готовых к выполнению. Эти слаги – фундаментальный аспект того, что происходит во время выполнения приложения. Они содержат скомпилированное, собранное приложение готовое к запуску вместе с инструкциями (Procfile) о возможности выполнения.

Heroku запускает приложение, выполняя команду, указанную вами в Procfile, на dyno, который был предварительно загружен с подготовленным слагом (фактически, с вашим релизом, который расширяет слаг) и некоторые еще не определенные элементы: config vars и add-ons.

Вместо управления оборудованием мы развертываем приложение в Heroku, которое упаковывает код и зависимости приложения в контейнеры – легкие, изолированные среды, обеспечивающие вычисления, предоставляя память, операционную систему и эфемерную файловую систему. Контейнеры обычно работают на общем хосте, но полностью изолированы друг от друга.

Как правило, если мы развертываем приложение в первый раз, Heroku автоматически запускает один веб-dyno, т.е. он загрузит dyno вместе со слагом и выполнит команду, связанную с типом веб-процесса в нашем Procfile. Мы можем контролировать, сколько dynos работает в любой момент времени.

Приложения обычно используют надстройки для предоставления вспомогательных услуг, таких как базы данных, системы очередей и кэширования, хранилища, службы электронной почты и многое другое. Heroku рассматривает эти надстройки как вложенные ресурсы: подготовка надстройки заключается в выборе одной из них на рынке и установке ее в приложение. Надстройки связаны с приложением, очень похожи на конфигурационные переменные, и поэтому необходимо уточнить раннее определение выпуска. Релиз приложений – это не просто наши настройки, это наш слаг, config vars, а также набор предоставляемых дополнений [3].

Дополнения Heroku – это компоненты, которые поддерживают приложение, такие как хранилище данных, мониторинг, аналитика, обработка данных и многое другое. В своем приложении будем использовать надстройку ClearDB, которая используется для работы с MySQL. ClearDB использует собственный MySQL для работы приложений, поэтому не нужно беспокоиться о какой-либо специальной обработке или преобразовании данных при работе с нашим приложением. ClearDB становится частью нашего стека приложений, как только приложение было помещено в Heroku. Даже если есть проблемы с сетью, сбои EBS, стихийные бедствия, наши данные останутся в сети и будут доступны.

Шифрование проходит везде: данные в покое зашифрованы, резервные копии сжимаются и шифруются, а сетевое шифрование доступно с использованием поддержки MySQL SSL. Кроме того, данные немедленно удаляются при удалении экземпляра надстройки из нашего приложения, поэтому можем быть уверены, что данные будут безопасны не только при их использовании, но и при их удалении. ClearDB также применяет строгий оперативный контроль доступа к данным, чтобы предотвратить доступ несанкционированного персонала к нашим данным без разрешения [4].

Так как ClearDB обеспечивает необходимую безопасность данных и связан напрямую с Heroku, было принято решение разместить свою базу данных именно там.

Развертывание базы данных в ClearDB осуществляются следующим образом:

1. Вначале надо зарегистрироваться на Heroku и развернуть там свое приложение.
2. Затем добавляем аддоны для своего приложения. Там выбираем аддон ClearDB и для него необходимый тариф, в зависимости от которого будем иметь различное количество доступных подключений и размер самой базы данных.
3. Heroku выдаст необходимый url базы данных для подключения и создания там своей базы.
4. Подключаемся с помощью MySQL к базе данных, импортируем туда необходимую базу даных.

В ходе работы с базой данных выяснилось, что если создать соединение к базе данных и не использовать его 60 секунд, то

плагин автоматически закроет эти соединения в связи с загруженностью своей системы. Поэтому было принято решение создать поток RenewConnectionThread, который будет обновлять соединения к базе данных каждые 30 секунд, чтобы пользователь мог оставить свою систему ненадолго и при попытке получить доступ к данным приложение не прекращало свое работу.

Физическая инфраструктура Heroku размещается и управляется в безопасных центрах данных Amazon и использует технологию Amazon Web Service (AWS). Amazon постоянно управляет рисками и регулярно проводит оценки для обеспечения соответствия отраслевым стандартам.

Стороннее тестирование безопасности приложения Heroku выполняется независимыми и авторитетными консалтинговыми фирмами. Результаты каждой оценки проверяются оценщиками, ранжируются по риску и назначаются ответственной команде.

Heroku использует изоляцию приложений, ограничения операционной системы и зашифрованные соединения для дальнейшего снижения рисков на всех уровнях [5].

Приложения на платформе Heroku работают в собственной изолированной среде и не могут взаимодействовать с другими приложениями или областями системы для предотвращения проблем безопасности и стабильности. Эти автономные среды изолируют процессы, память и файловую систему, в то время как брандмауэры на основе хоста не позволяют приложениям устанавливать локальные сетевые подключения.

Структура и функционирование приложения

Разработанное приложение предназначено для информационного обеспечения учебного процесса. Проектирование и безопасность приложения поддерживается инструментальными средствами, представленными в п. 2.

Чтобы ролевая система работала, роль пользователя постоянно хранится в объекте пользователя, который лежит в данной сессии. При обращении к каждой странице происходит проверка роли пользователя на соответствие после чего он направляется на главную страницу приложения.

Предложены следующие роли и разработан функционал для них: студент; преподаватель; администрация факультета; администрация университета; гость.

Для серверной части нами была выбрана мультиплатформенная Java EE(Enterprise Edition), так как это мультиплатформенный язык, а такие веб приложения поддерживаются большим количеством различных сервисов. Приложение строится следующим образом: есть jsp(Java Servlet Page) и main servlet. Со стороны jsp будут поступать запросы на сервер, которые адресуются сервлету. Сервлет распознает запрос, выполняет необходимые действия и возвращает страницу jsp ответа.

Проект собирается при помощи Apache Maven – фреймворка для автоматизации сборки проектов на основе описания их структуры в файлах на языке POM, являющимся подмножеством XML. В файле pom.xml указываются зависимости проектов, версии различных библиотек, необходимых для работы приложения. И каждый раз при сборке проекта библиотеки подтягиваются к проекту с помощью данного файла. После сборки проекта получаем файл с расширением .war, который содержит в себе все скомпилированные классы и все необходимые ресурсы приложения. Запускается такое приложение при помощи контейнера сервлетов Tomcat.

Для работы с базами данных использовался СУБД MySql [6] и надстройка ClearDB. Соответственно в Java использовался JDBC. Все основные сущности в JDBC API, с которыми предстоит работать, являются интерфейсами: Connection; Statement; PreparedStatement; CallableStatement; ResultSet; Driver; DatabaseMetaData.

Вначале вызываем метод forName для создания класса драйвера. Далее у объекта драйвера используя property, которые содержат данные о нашем соединении с базой данных, создаем Connection. Для этого Connection создаются и выполняются запросы.

Развертка приложения происходит на Heroku, поддерживающей ряд языков программирования.

Схема базы данных (рисунок 1) содержит таблицы:

- administration (данные о деканате);
- group_subject (какие предметы ведет какой преподаватель);
- mark(оценки в семестре);
- rectorate(данные о ректорате);
- role(типы пользователей);
- session_mark(оценки за сессию);
- session_mark_type(типы оценок за сессию);
- student(данные о студентах);
- subject(данные о предметах);
- subject_category(категории предмета, например лабораторные);
- subject_date(даты занятий по предметам);
- teacher(данные о преподавателях);
- user(данные о пользователях).

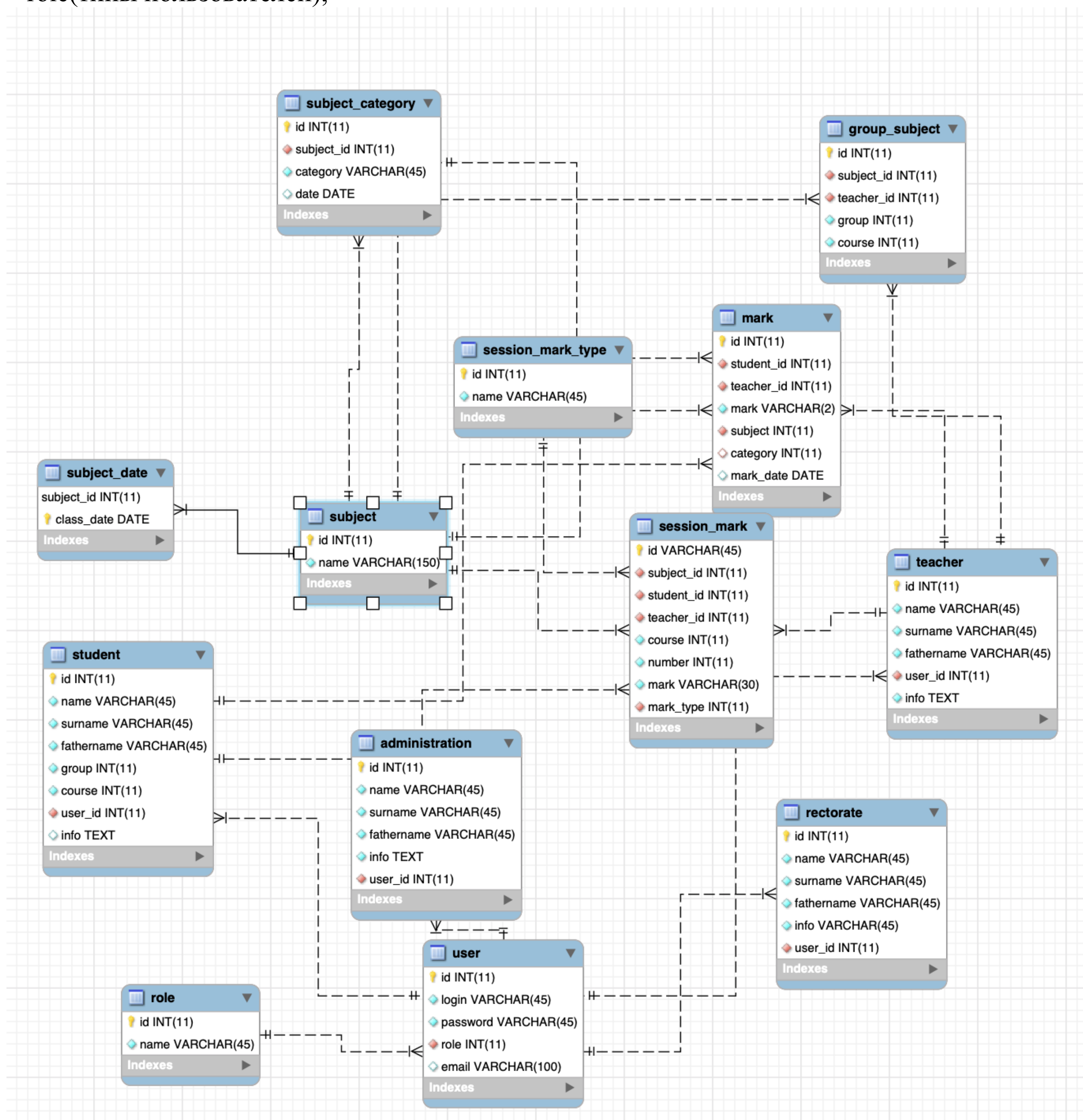


Рис. 1. Схема базы данных приложения

Есть общая таблица User, где хранятся основные данные о пользователе, такие как его логин, пароль и роль в системе, а также набор дополнительных параметров пользователя, таких как электронная почта, № телефона и т.д. Таблицы administration, rectorate, student и teacher хранят в себе id своего пользователя, причем пользователь может относиться к одной из своих ролей. Таблица subject содержит набор названий предметов, а в таблице subject_category, хранятся результаты промежуточного контроля знаний по предмету и subject_date с датами занятий по предмету. Данная таблица связывается с таблицей teacher с помощью промежуточной таблицы group_subject, где хранятся id преподавателя, id предмета, группа и курс, у которого преподаватель ведет данный предмет.

Работа веб-приложения осуществляется следующим образом: пользователь выбирает какое-то действие на странице, действие поступает на сервер, принимая решение, каким образом его обработать, и возвращает ответ пользователю. Пользователь делает какое-то действие на странице, запрос отправляется на сервер и обрабатывается классом MainController – наследником HttpSerlvet. Для обработки запросов в сервлете используется паттерн Command, в котором объект, вызывающий операцию, отделяется от объекта, знающего, как эту операцию выполнить.

В нашей реализации в интерфейсе Command используется метод execute. Он принимает на вход параметр request, а на выходе выдает объект типа CommandType, который состоит из типа перехода: Redirect, Forward, и страницы, на которую перейдет пользователь после выполнения своего действия. Для работы с базой данных используется шаблон DAO (Data Access Object) [7].

Для безопасной передачи пароля через запросы используется класс MessageDigest. Пароли хранятся в базе данных в виде хэш-функций. Пароль, поступающий из запроса, преобразуется функцией и сравнивается с соответствующим полем базы данных [8]. Для отображения информации использовалась технология JSP (Java Servlet Pages), позволяющая веб-разработчикам и дизайнерам быстро разрабатывать и легко поддерживать насыщенные информацией динамические веб-страницы, которые используют существующие бизнес-системы. JSP обеспечивает быструю разработку веб-приложений, не зависящих от платформы.

Технология JSP и сервлеты предоставляют привлекательную альтернативу другим типам динамических веб-сценариев/программ, предлагая независимость от платформы, улучшенную производительность, отделение логики от дисплея, простоту администрирования и использования [9].

Для авторизации пользователь заходит на главную страницу, т.е. выполняет роль гостя. Затем он попадает на страницу авторизации, так он не имеет какой-то конкретной роли.

Пользователь вводит логин и пароль (рисунок 2) и снова попадает на главную страницу. Пароли в базе данных хранятся в зашифрованном виде, так что перед тем, как передать пароль далее его необходимо зашифровать таким же образом, как и в базе данных. Пароль шифруется с помощью кастомного класса Encryptor, в котором используется шифрование MD5 с помощью класса MessageDigest и возвращается обратно в команду. Далее происходит обращение к сервису к методу public User checkUser(String login, String password), который возвращает пользователя, если такой существует, либо null в ином случае. После авторизации пользователю присваивается роль, и в зависимости от ее определяется функционал, в рамках которого пользователь может производить фиксированные действия в приложении.

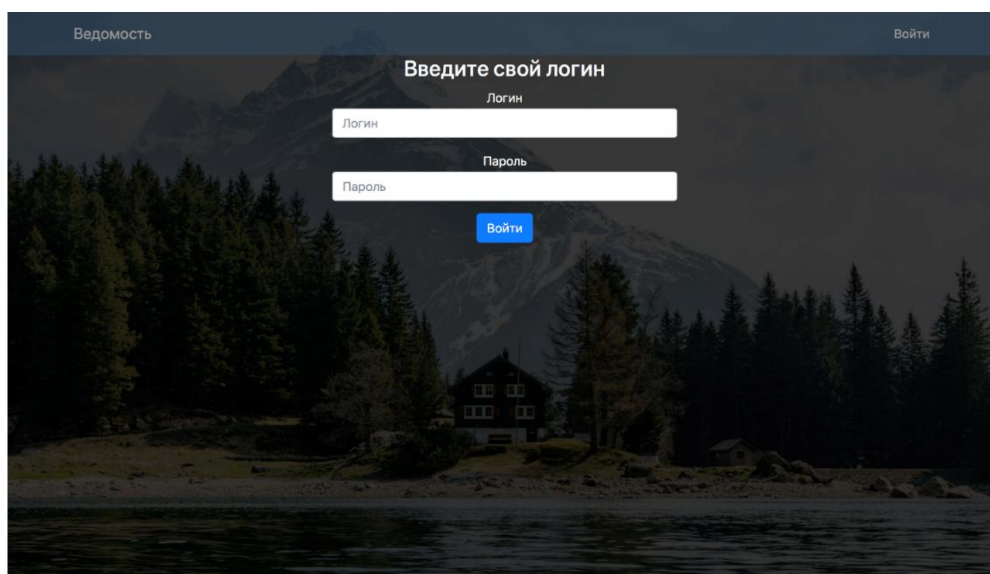


Рис. 2. Страница авторизации

Студент видит только те предметы, которые на данный момент у него имеются и только информацию, относящуюся к его группе. Осуществляется это за счет того, что в базе данных хранятся предметы и их отношение к определенному курсу и группе. Информацию же о курсе и группе можно получить из объекта студента, который находится внутри объекта пользователя.

Пользователь выбирает необходимый ему предмет и оказывается на странице предмета. Там отражается список группы с посещаемостью, курс и номер данной группы, название предмета, а также имя преподавателя. Студент может посмотреть оценки своей группы по данному предмету. Студент в ведомости не может что-либо изменить, ему лишь доступна возможность просмотра информации. Он может просмотреть свои текущие оценки за семестр, а также за предыдущие сессии.

У преподавателя кроме возможности посмотреть оценки обучающихся по данному предмету есть возможность изменять, удалять либо добавлять оценки, а также изменять или добавлять какие-либо категории, например, даты посещаемости, лабораторные работы, контрольные работы, экзамен, рейтинг и т.д. (рисунок 3).

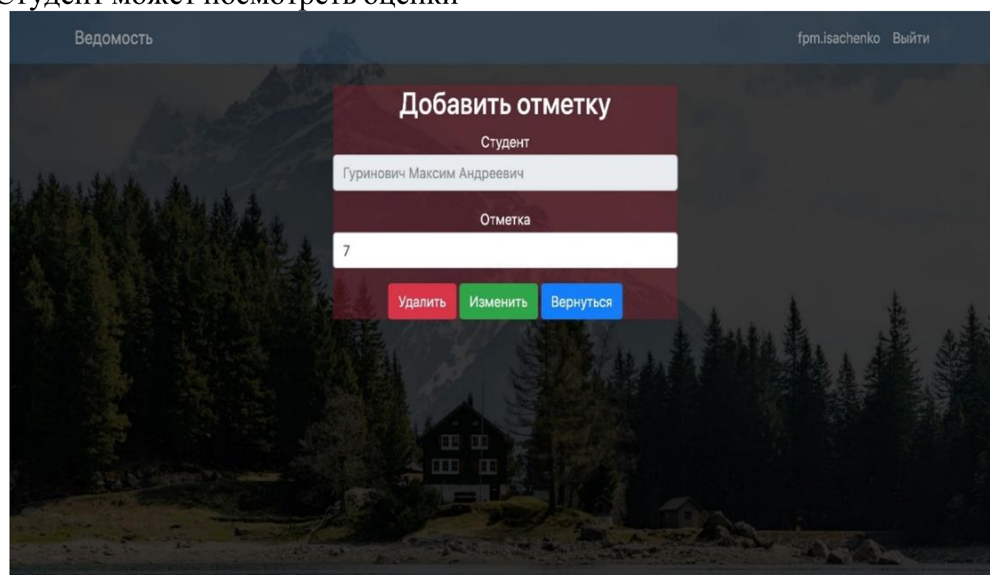


Рис. 3. Страница редактирования оценки

Сотрудники деканата могут просматривать всю информацию о студентах факультета и при необходимости вносить необходимые изменения.

Имеется возможность уведомления студента о решениях деканата.

Возможности ректората включают дополнительно запросы к деканату о происходящих событиях. Приложение автоматически закрывается после одной минуты простаивания.

На данный момент для приложения используется бесплатный тариф Heroku. В зависимости от выбранного тарифа возможности по времени и используемой памяти различны.

Созданное приложение существенно упрощает процесс перевода обучающихся с одного уровня образования на другой [10].

Заключение

Разработанное приложение позволяет осуществить цифровую трансформацию информационного обеспечения учебного процесса. Результаты работы приложения дают возможность оптимизировать процесс взаимодействия различных структур учебного заведения. Существенно упрощается процесс перевода студентов вверх /вниз по уровням обучения.

Литература

1. What is cloud computing? (2018). URL: https://www.salesforce.com/what-is-cloud-computing
2. Что такое облачные сервисы? (2012). URL: https://www.moysklad.ru/poleznoe/statyi/chto-takoe-oblachnye-servisy
3. How Heroku Works (2019). Центр разработки Heroku. URL: https://devcenter.heroku.com/articles/how-heroku-works
4. Документация ClearDB (2019). Центр разработки Heroku. URL: https://devcenter.heroku.com/articles/cleardb
5. Heroku Security (2019). URL: https://www.heroku.com/policy/security
6. Гольцман, В.Г. (2010). MySQL 5.0. Библиотека программиста. Питер.
7. Блинов, И.Н. (2007). Java. Промышленное программирование. УниверсалПресс.
8. Class MessageDigest (2018). Документация Oracle. URL: https://docs.oracle.com/javase/7/docs/api/java/security/MessageDigest.html.
9. JavaServer Pages Technology (2019). Oracle. URL: https://www.oracle.com/technetwork/java/javaee/jsp/index.html.
10. Буза, М.К. (2019). Подготовка квалифицированных IT-специалистов в условиях цифровой трансформации образования. *Цифровая трансформация*, №1(6), 81-84.

References

1. What is cloud computing? (2018). URL: https://www.salesforce.com/what-is-cloud-computing.
2. Chto takoe oblachnye servisy? (2012). URL: https://www.moysklad.ru/poleznoe/statyi/chto-takoe-oblachnye-servisy
3. How Heroku Works (2019). Tsentr razrabotki Heroku. URL: https://devcenter.heroku.com/articles/how-heroku-works
4. Dokumentatsiya ClearDB (2019). Tsentr razrabotki Heroku. URL: https://devcenter.heroku.com/articles/cleardb
5. Heroku Security (2019). URL: https://www.heroku.com/policy/security
6. Goltsman, V.G. (2010). MySQL 5.0. Biblioteka programmista. Piter.
7. Blinov, I.N. (2007). Java. Promyshlennoe programmirovaniye. UniversalPress.
8. Class MessageDigest (2018). Dokumentatsiya Oracle. URL: https://docs.oracle.com/javase/7/docs/api/java/security/MessageDigest.html.
9. JavaServer Pages Technology (2019). Oracle. URL: https://www.oracle.com/technetwork/java/javaee/jsp/index.html.
10. Buza. M.K. (2019). Podgotovka kvalifitsirovannykh IT-spetsialistov v usloviyakh tsifrovoy transformatsii obrazovaniya. Tsifrovaya transformatsiya. №1(6). 81-84.

RESUME

M.K. Bouza

Cloud technology is the foundation for designing efficient applications

In connection with the transition to the digital economy, the article presents developed application of information support for the educational process. Digital technologies require the placement of data, algorithms and software using cloud technology. The tool environment for the design of this application was

formed and argued, including Heroku, ClearDB, MySQL SSL, Apache Maven. Various models of cloud technologies are analyzed and SaaS technology is selected and argued for the solution of the task.

Most SaaS applications run directly through a web browser and do not require downloads or installations and or installations on the client side. SaaS vendors solve all the technical problems associated with data storage.

To support cloud applications, redistribute data between them, as well as ensure information security, the free Heroku service was chosen. Heroku supports a wide range of programming languages, and launching them in Heroku does not require changes to applications, it allows you to create, run and scale applications in the same way in all languages.

To deploy the application, a Git-distributed version control system was used, as well as ClearDB for storing and working with the database.

Web applications created with the support of selected tools and allows you to create and maintain an information component: academic records, reports, community service, promotion and message distribution.

Server requests come from the JSP (Java Server Page) and are addressed to the servlet. The servlet recognizes the request, performs the necessary actions, and returns the jsp response page.

Application users are divided into clusters. The role functions of users of each cluster are formed and supported, which allows you to effectively and quickly respond to various problems in the learning process. The “student-teacher-dean-rector’s” chain is maintained through the distribution of messa-ges. In a multi-level education system, the infor-mation component allows you to quickly move students to levels throughout the school year.

Надійшла до редакції 15.05.2019